Guy Barry / Martin Pickering

Dependency and Constituency in Categorial Grammar

Word Order in Categorial Grammar / L'ordre des mots dans les grammaires catégorielles. Proceedings of the workshop of Clermont-Ferrand on behalf of DYANA project (May 25-27 1990) /Actes du colloque de Clermont-Ferrand tenu dans le cadre du projet DYANA (25-27 mai 1990), edited by / publiés par Alain Lecomte. Clermont-Ferrand, éditions Adosa, 1992. Pp. 39-57.

ABSTRACT

Unlike traditional notions of constituency, 'flexible constituency' allows two constituents to overlap. In this paper we develop a particular system of flexible constituency within categorial grammar. We first discuss the notion of dependency, and show how the categorial grammar formalism may be used to formulate the concepts of 'head' and 'dependent'. From this we define *dependency constituency*, where a dependency constituent corresponds to a string of words connected by dependencies. We then develop an account of coordination based on dependency constituency, which claims that strings can be coordinated if and only if they consist of sequences of dependency constituents of the same types in the same order.

Dépendance et Constituence dans une Grammaire Catégorielle

RÉSUMÉ

A la différence des notions traditionnelles de constituence, la "constituence flexible" permet la superposition de deux constituants. Dans cette communication, nous développons un système spécifique de constituence flexible dans une grammaire catégorielle. Nous discutons d'abord la notion de dépendance, et nous montrons comment le formalisme de grammaire catégorielle peut être utilisé pour formuler les concepts de "tête" et de "dépendant". A partir de là, nous définissons la "constituence de dépendance", où un constituent de dépendance correspond à une chaîne de mots liés au moyen de dépendances. Nous développons ensuite une explication de la coordination basée sur la constituence de dépendance, selon laquelle les chaînes peuvent être coordonnées si et seulement si elles sont composées de chaînes de constituents de dépendance des mêmes types et dans le même ordre.

Guy Barry, Martin Pickering
University of Edinburgh
Centre for Cognitive Science
2 Buccleuch Place
Edinburgh EH8 9LW, Scotland
(United Kingdom).

1 Introduction

It has been seen as an advantage of some systems of categorial grammar (Steedman 1987; Moortgat 1988) that they allow the definition of some notion of 'flexible constituency'. This has been argued to allow accounts of syntactic phenomena such as coordination and extraction, as well as other phenomena such as incremental interpretation and intonational structure.

In phrase-structure theories of grammar, category symbols are merely labels for constituents, and so by definition everything that can be given a category must be a constituent. In categorial grammars, on the other hand, the symbols used reflect more general combining properties of strings. To avoid confusion we shall use the word ***type*** rather than ***category*** to refer to these symbols. Nevertheless, there is still an implicit assumption that any string that can be given a type is a constituent. We shall refer to this assumption as the ***type-constituent correspondence*** hypothesis.

In the Lambek calculus, every string can be given a type, so type-constituent correspondence would predict that every string is a constituent. If the notion of constituency has any independent linguistic use, then we cannot maintain type-constituent correspondence within the Lambek calculus. This indicates that we must either abandon the Lambek calculus as a linguistic framework or abandon type-constituent correspondence.

The former approach suggests the use of a more restricted calculus such as Steedman's Combinatory Categorial Grammar (CCG). By means of a particular set of type-combining rules, Steedman attempts to assign types to all and only those strings that can be regarded as constituents. This approach is open to two criticisms. Firstly, the type-combining rules are proposed on the basis of specific linguistic phenomena rather than general principles, so that the set of constituents generated is in some sense arbitrary, and leads to some anomalies in linguistic description. This could perhaps however be rectified by a more principled choice of rules. Secondly, though, Steedman's approach has no way of characterizing strings that are not constituents, even though it appears (as we shall argue) that the characterization of such strings is necessary for the description of certain linguistic phenomena.

In this paper we shall investigate the alternative approach of retaining the general Lambek framework, but dropping type-constituent correspondence. Instead we shall argue from the standpoint of dependency, by proposing a definition of dependency within the Lambek calculus, and from it deriving a new notion of constituency, which we shall refer to as ***dependency constituency***. We shall argue that the notion of a dependency constituent has applications in the syntactic description of coordinate constructions.

2 Dependencies and Dependency Constituents

2.1 Heads and Dependents

All notions of dependency rely on a primitive notion of *head*; roughly speaking, the head is the element on which other elements depend. For example, consider the following sentence:

(1) Bill talks to Mary.

Three approaches seem plausible. If we assume that heads and dependents are both phrases, then given standard phrase-structure assumptions *talks to Mary* is the head of the sentence and has the single dependent *Bill*, and *talks* is the head of *talks to Mary* and has the single dependent *to Mary*. Alternatively, if we assume that heads are words and dependents are phrases, then *talks* can be taken as the head of the sentence and as having the two dependents *to Mary* and *Bill*. Finally, if we assume that heads and dependents are both words, then *talks* can be taken as the head of the sentence and as having the two dependents *to* and *Bill*, and *to* can be taken as having the single dependent *Mary*.

There is no primitive notion of head in categorial grammar, but the notion of semantic/syntactic *functor* is often taken to be head-like. This raises two questions; should categorial grammar be thought of as having phrasal or lexical heads and dependents, and what is the relationship of functors to 'traditional' (linguistically motivated) heads?

To answer the first point, let us limit our attention for the moment to applicative categorial grammar (**AB**). **AB** can be thought of as having phrasal heads and phrasal dependents if we adopt the following definition (where the semantic type $Y \to X$ collapses the two syntactic types X/Y, $Y\backslash X$):

Definition 1 *When a string of a functor type* $Y \to X$ *combines with a string of an argument type* Y *to form a string of type* X*, the functor string is the* phrasal head *and the argument string the* (phrasal) dependent.

Thus in *Bill talks to Mary* of type S, *talks to Mary* of type NP\S is the phrasal head, and *Bill* of type NP is the dependent. Similarly, in *talks to Mary* of type NP\S, *talks* of type (NP\S)/PP is the phrasal head, and *to Mary* of type PP is the dependent.

But we may also regard **AB** as having lexical heads and phrasal dependents if we adopt the following alternative definition:

Definition 2 *When a word of a functor type* $Y_1 \to (\ldots \to (Y_n \to X)\ldots)$ *combines with strings of argument types* $Y_1, \ldots, Y_n$ *to form a string of type* X*, the functor word is the* lexical head *and the argument strings the* (phrasal) dependents.

Thus, in *Bill talks to Mary* of type S, *talks* of type (NP\S)/PP is the lexical head, and *to Mary* of type PP and *Bill* of type NP are the dependents.

In other words, we can derive the notion of a lexical head from that of a phrasal head by a process similar to 'uncurrying' of functions. But it seems more natural to view dependents as phrasal, since functions take expressions of phrasal rather than lexical types as arguments. (This view will be reinforced when we consider the full Lambek calculus below.)

The second point begs the question of what is the traditionally accepted notion of head, about which there is widespread disagreement. We shall not go into the issues here; see Zwicky (1985) and Hudson's (1987) reply to Zwicky for a comprehensive discussion. Hudson gives a great deal of evidence which suggests that, for constructions involving complementation, most syntactically motivated definitions of head in fact coincide with the notion of semantic functor. However, the reverse appears to be true in modifier constructions, where according to Hudson's criteria for headship (and nearly all linguistic theories) the modified element appears to be the head, whereas on semantic grounds the modifier is standardly assumed to be the functor (as witnessed by the most common categorial type assignments N/N, N\N, (NP\S)\(NP\S) for adjectives, relative clauses and adverbs respectively).

There are three possible solutions to this problem, none of them entirely satisfactory. We could deny the equivalence of head and functor, but this would require us to define a separate theoretical primitive of 'head' independently of the categorial grammar formalism; we shall not pursue this further. Alternatively we could claim that modifiers are heads, although as we shall see in subsection 2.3 this approach leads to an unsatisfactory notion of constituency. Finally we could claim that modified elements are functors, which seems to be supported by the fact that the distinction between modifiers and optional arguments is not always clear.

We shall formalize the above description of modifiers by assuming that modifiers have atomic types, and that modifiable types are functions over zero or more premodifiers and zero or more postmodifiers (cf. HPSG (Pollard and Sag 1987)). We can achieve this by extending the categorial machinery with a 'Kleene star' structural operator, so that

X^* means 'zero or more occurrences of X'.[1] We shall classify noun premodifiers as AdnPre, noun postmodifiers as AdnPost, verb premodifiers as AdvPre and verb postmodifiers as AdvPost. Thus lexical nouns will be categorized as

$$(\text{AdnPre}^*\backslash \text{N})/\text{AdnPost}^*,$$

from which the types N, AdnPre\N, AdnPre\(AdnPre\N), N/AdnPost etc. are derivable. Similarly intransitive verbs will be categorized as

$$(\text{AdvPre}^*\backslash(\text{NP}\backslash \text{S}))/\text{AdvPost}^*,$$

transitive verbs as

$$((\text{AdvPre}^*\backslash(\text{NP}\backslash \text{S}))/\text{AdvPost}^*)/\text{NP},$$

and so on. Since it would be unwieldy (and uninformative) to write down these complex types in every derivation, we shall usually write only the derived type that is appropriate to the particular derivation. For example, *man* will be given the type N in *the man walks*, AdnPre\N in *the old man walks*, N/AdnPost in *the man who I like walks* and so on.

2.2 Dependency in the Lambek Calculus

When we try to generalize the ideas of head and dependent developed above for **AB** to the full Lambek calculus **L**, we run into a problem, since the roles of functor and argument can be reversed. Suppose we assume that head-dependent relations are based on the functor-argument relations implicit in lexical type assignments. Then, for instance, *John walks* consists of a function *walks* of type NP\S applying to an argument *John* of type NP, and so *walks* is the head. But if we assign the higher type S/(NP\S) to *John*, then *John* appears to be the head. If the basic meaning of *John* is represented as **john** in the first case, then its meaning in the second case will be $\lambda f.f$ **john**, where f is a variable of type $\text{NP}\rightarrow\text{S}$; when *John* is type-raised, the dependency relations implicit in the lexical assignments are lost. We shall refer to a meaning representation as *dependency-preserving* if it maintains the head-dependent relations derivable from lexical assignments. More formally:

Definition 3 *A lambda-calculus meaning representation is* dependency-preserving *iff it does not involve abstraction of a variable that occurs as a functor within it.*

As pointed out in Morrill, Leslie, Hepple and Barry (1990), there is a direct correspondence between such lambda-terms and derivations in the 'natural deduction' style formulation of **L**. Thus definition 3 is equivalent to the following:

Definition 4 *A derivation in* **L** *is* dependency-preserving *iff it does not discharge an assumption that forms the major premise of an Elimination inference.*

So for example consider the six derivations below, and their associated lambda-terms. The three in (2) are dependency-preserving, and the three in (3) are not:[2]

(2) a.

$$\dfrac{\dfrac{\textbf{the}}{\text{NP/N}} \quad \dfrac{\textbf{dog}}{\text{N}}}{\text{NP}}\,/E$$

the dog

b.

$$\dfrac{\dfrac{\dfrac{\textbf{John}}{\text{NP}} \quad \dfrac{\dfrac{\textbf{likes}}{(\text{NP}\backslash\text{S})/\text{NP}} \quad [\text{NP}]_1}{\text{NP}\backslash\text{S}}\,/E}{\text{S}}\,\backslash E}{\text{S/NP}}\,/I_1$$

λx^{NP}[**likes** x **john**]

c.

$$\dfrac{\dfrac{\dfrac{\textbf{will}}{(\text{NP}\backslash\text{S})/\text{VP}} \quad \dfrac{\dfrac{\textbf{see}}{\text{VP/NP}} \quad [\text{NP}]_1}{\text{VP}}\,/E}{\text{NP}\backslash\text{S}}\,/E}{(\text{NP}\backslash\text{S})/\text{NP}}\,/I_1$$

λx^{NP}[**will** (**see** x)]

(3) a.

$$\dfrac{\dfrac{\dfrac{[\text{NP/N}]_1 \quad \dfrac{\textbf{dog}}{\text{N}}}{\text{NP}}\,/E \quad \dfrac{\textbf{runs}}{\text{NP}\backslash\text{S}}}{\text{S}}\,\backslash E}{(\text{NP/N})\backslash\text{S}}\,\backslash I_1$$

$\lambda f^{\text{N}\rightarrow\text{NP}}$[**runs** ($f$ **dog**)]

b.

$$\dfrac{\dfrac{\dfrac{\textbf{that}}{\text{SP/S}} \quad \dfrac{\dfrac{\textbf{Harry}}{\text{NP}} \quad [\text{NP}\backslash\text{S}]_1}{\text{S}}\,\backslash E}{\text{SP}}\,/E}{\text{SP}/(\text{NP}\backslash\text{S})}\,/I_1$$

$\lambda f^{\text{NP}\rightarrow\text{S}}$[**that** ($f$ **harry**)]

c.

$$\dfrac{\dfrac{\dfrac{[((\text{NP}\backslash\text{S})/\text{NP})/\text{NP}]_1 \quad \dfrac{\textbf{Mary}}{\text{NP}}}{(\text{NP}\backslash\text{S})/\text{NP}}\,/E \quad \dfrac{\textbf{John}}{\text{NP}}}{\text{NP}\backslash\text{S}}\,/E}{(((\text{NP}\backslash\text{S})/\text{NP})/\text{NP})\backslash(\text{NP}\backslash\text{S})}\,\backslash I_1$$

$\lambda f^{\text{NP}\rightarrow(\text{NP}\rightarrow(\text{NP}\rightarrow\text{S}))}$[$f$ **mary john**]

The term **the dog** is dependency-preserving because it does not involve abstraction, and the terms λx[**likes** x **john**] and λx[**will** (**see** x)] are dependency-preserving because in each case the abstracted variable does not occur as a functor. On the other hand, the terms λf[**runs** (f **dog**)], λf[**that** (f **harry**)] and λf[f **mary john**] are not dependency-preserving, since in each case the abstracted variable is a functor over one or more arguments.

We shall refer to strings with dependency-preserving analyses as *dependency constituents.* More precisely:

Definition 5 *A* dependency constituent *is a string (under a particular reading) whose normal-form derivation in* **L** *is dependency-preserving (equivalently, for which* some *derivation in* **L** *is dependency-preserving).*

(The equivalence of the two definitions is immediate because the normalization process never introduces new functors.) Thus the underlined substrings in (4) below are analysable as dependency constituents, whereas those in (5) are not:

(4) a. The dog runs.
    b. John likes Mary.
    c. John will see Mary.

(5) a. The dog runs.
    b. I think that Harry left.
    c. I showed Mary John.

Intuitively, we may think of the underlined substrings in (5) as having 'missing heads'; two words cannot be related because the head of one or both of them is absent from the string.

If we take the traditional notion of constituent to correspond to strings with purely applicative derivations, then the notion of dependency constituent is seen to subsume the traditional notion. The advantage of this notion of constituency is that it allows a degree of flexibility in constituent structure while still requiring that constituents are (in some sense) semantically coherent units. For example, consider:

(6) John thinks that Harry likes Mary.

Here, the substrings *that Harry*, *thinks that Harry* and *John thinks that Harry* are not dependency constituents, but every other substring (including one-word strings and the entire sentence) is, e.g. *John thinks, John thinks that, that Harry likes, thinks that Harry likes.*

2.3 Consequences of Dependency Constituency

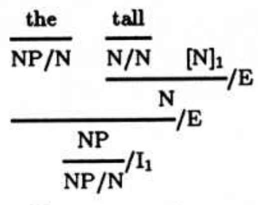
Clearly the choice of head in modifier constructions will make a difference to what strings are regarded as dependency constituents. Consider for example:

(7) the tall man

(8) the man who I saw

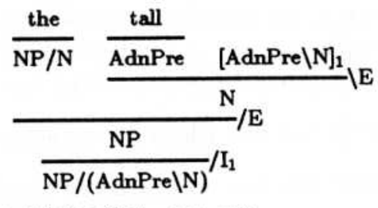
If modifiers are regarded as heads, the underlined substring in (7) will be a dependency constituent, but the one in (8) will not be. On the other hand, if modified elements are regarded as heads, the underlined substring in (7) will not be a dependency constituent, but the one in (8) will be. The relevant proofs and lambda-terms follow:

(9) a.



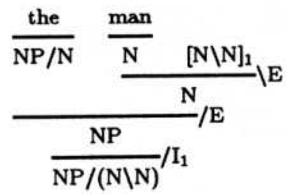
λx^{N}[the (tall x)]

b.



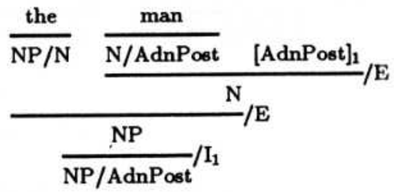
$\lambda f^{AdnPre \to N}$[the (f tall)]

(10) a.



$\lambda f^{N \to N}$[the (f man)]

b.



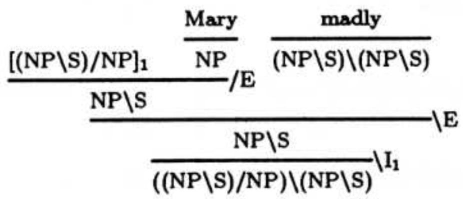
$\lambda x^{AdnPost}$[the (man x)]

This would seem to support the choice of modified element as head, since there is a clear intuitive connection between *the* and *man*, but not between *the* and *tall*. We shall assume modified elements to be heads for the rest of the paper (though we shall sometimes give alternative analyses where modifiers are assumed to be heads).

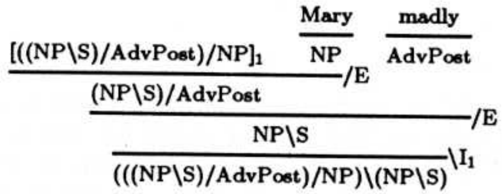
It is worth noting here that the string underlined in (11) cannot be a dependency constituent, whatever the choice of head in modifier constructions:

(11) John loves Mary madly.

(12) a.



$\lambda f^{NP \to (NP \to S)}$[madly ($f$ mary)] b.



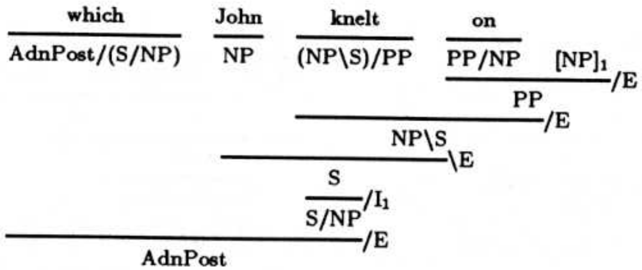
$\lambda f^{NP \rightarrow (AdvPost \rightarrow (NP \rightarrow S))}$[($f$ **mary**) **madly**]

Some interesting consequences of this definition of dependency emerge when we consider constructions involving extraction. Firstly, as one might expect, any relative clause is a dependency constituent, e.g. the underlined fragments of the following two constructions:[3]

(13) the box <u>which John knelt on</u>

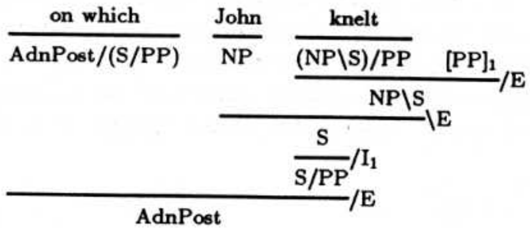
(14) the box <u>on which John knelt</u>

(15)



which (λx^{NP}[**knelt** (**on** x) **john**])

(16)



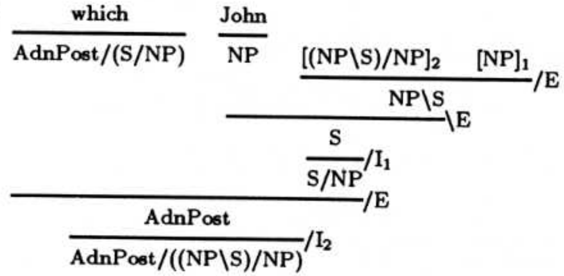
on-which (λx^{PP}[**knelt** x **john**])

On the other hand, some fragments of relative clauses are not dependency constituents, such as the two underlined below:

(17) the box <u>which John</u> knelt on

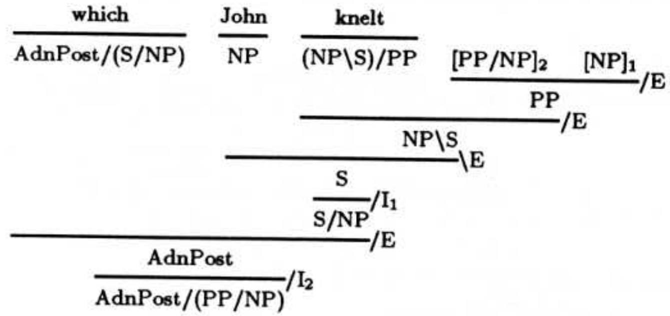
(18) the box <u>which John knelt</u> on

(19)



$\lambda f^{NP \rightarrow (NP \rightarrow S)}$[**which** ($\lambda x^{NP}$[$f$ x **john**])]

(20)



$\lambda g^{NP \rightarrow PP}$[**which** ($\lambda x^{NP}$[**knelt** ($g$ x) **john**])]

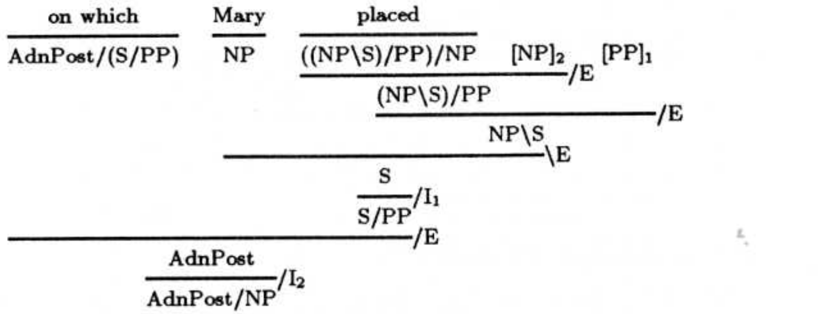
By our definition, neither derivation is dependency-preserving, since in each case an abstracted variable acts as a functor ($f^{NP \rightarrow (NP \rightarrow S)}$ in the first case, $g^{NP \rightarrow PP}$ in the second case). But each time the argument of the abstracted functor is not a constant but another abstracted variable, so that the 'headless' element is not a lexical item.

Thus we see that definition 3 enables us to make some subtle distinctions. For example, the underlined phrase in (21) will be analysed as a dependency constituent, whereas that in (22) will not:

(21) the tray <u>on which Mary placed</u> the cake

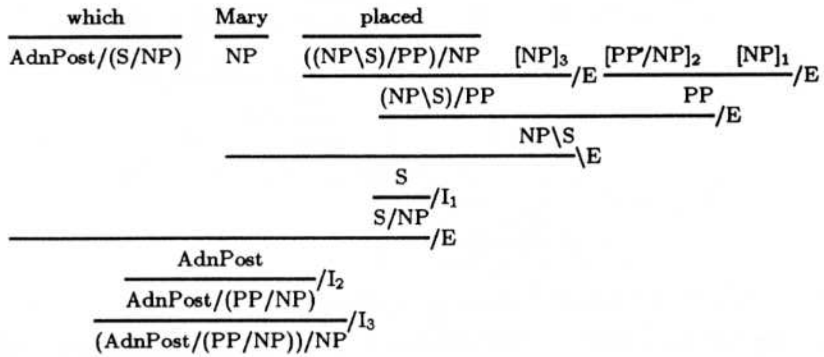
(22) the tray <u>which Mary placed</u> the cake on

(23)



$\lambda y^{NP}[\textbf{on-which}\ (\lambda x^{PP}[\textbf{placed}\ y\ x\ \textbf{mary}])]$

(24)



$\lambda y^{NP}\lambda g^{NP \rightarrow PP}[\textbf{which}\ (\lambda x^{NP}[\textbf{placed}\ y\ (g\ x)\ \textbf{mary}])]$

In the first case only complete arguments of *placed* are abstracted, but in the second the variable g is abstracted, which is itself a functor over the abstracted variable x. This seems to capture the intuition that in (21) the relative pronoun *on which* fills an argument role of the verb *placed*, whereas in (22) the relative pronoun *which* fills an argument role not of *placed* itself but of the prepositional argument of *placed* (which is outside the string).

To summarize this section, we have attempted to capture the intuitive notion of dependency within the Lambek calculus, and from it derived a flexible notion of constituency. We have thus abandoned the assumption that any string that can be assigned a type must be a constituent. Let us now see how this notion can be applied to problems of coordination.

3 Coordination

3.1 Coordination of Dependency Constituents

A claim often made in favour of flexible categorial grammars is their ability to deal with a large proportion of coordination phenomena by means of a single principle, usually assumed to be as follows:

Hypothesis 1 *Two (or more) strings may be coordinated to give a string of type X iff each has type X.*

For the purposes of this discussion we shall treat the two directions of implication in hypothesis 1 as two separate hypotheses. One direction may be phrased as follows:

Hypothesis 2 *If two (or more) strings can be given the same type X, then they can be coordinated to give a string of type X.*

We shall discuss the converse later (hypothesis 4).

The precise mechanism used to implement hypothesis 2 varies from formulation to formulation (e.g. polymorphic types for conjunctions, syncategorematic rule schemas), but the principle remains the same. In this section we are concerned not with the mechanism for coordination, but with the characterization of what strings can be coordinated. To save space in examples, instead of giving full derivations we shall usually merely bracket each conjunct, and specify the type that must be assigned to each conjunct (and to the coordinate structure) for the derivation to proceed.

If we are working within the full Lambek calculus, hypothesis 2 holds true in a large class of cases. For instance, we can clearly coordinate strings that form standard phrase structure constituents, like (25) below:

(25) John [sang some songs] and [played the piano].
Type of each conjunct: NP\S

We can also deal with cases like (26) and (27), both of which are usually classified under the heading of 'non-constituent' coordination:

(26) John [will buy] and [may eat] the beans.
Type of each conjunct: (NP\S)/NP

(27) John loves [Mary madly] and [Sue passionately].
Type of each conjunct: (((NP\S)/AdvPost*)/NP)\(NP\S)

In our terms, (25) and (26) both involve coordination of dependency constituents, while (27) involves coordination of non-dependency constituents. Although these three examples look very different, in each case the structure of each conjunct is the same, in the sense that each consists of a sequence of words of the same lexical types. Hypothesis 2 will always allow coordination in such cases, since **L** can always give the same type to two strings with the same structure (in this sense).

More interesting are cases where the conjuncts have different structures, such as (28) and (29):

(28) John [plays the piano] and [sings].
Type of each conjunct: NP\S

(29) John [bought] and [may eat] the beans.
Type of each conjunct: (NP\S)/NP

However, the unacceptability of (30) shows that hypothesis 2 overgenerates:[4]

(30) *John loves [Mary madly] and [Sue].
Type of each conjunct: (((NP\S)/AdvPost*)/NP)\(NP\S)

In (28) and (29) both conjuncts are again dependency constituents, but in (30) the first conjunct is not. Note that even if we took modifiers to be heads we could still give the conjuncts in (30) the shared type ((NP\S)/NP)\(NP\S).

In general, the restrictions on coordination of unlike structures seem to be stronger than those on coordination of like structures. For example, (31) appears to be acceptable, but (as noted by Steedman) (32) does not:

(31) [I believe that John] and [Harry thinks that Mary] is a genius.
Type of each conjunct: S/(NP\S)

(32) *[I believe that John] and [Mary] is a genius.
Type of each conjunct: S/(NP\S)

Note once more that both conjuncts in (31) and the first conjunct is (32) are not dependency constituents. Similarly, (33) seems acceptable, but not (34):

(33) [two small] and [three large] oranges
Type of each conjunct: NP/(AdnPre*\N)

(34) *[two small] and [three] oranges
Type of each conjunct: NP/(AdnPre*\N)

Here, the conjuncts in (33) and the first conjunct in (34) are not dependency constituents if modified elements are analysed as heads (although they would be if modifiers were analysed as heads).

This evidence suggests that there is some correlation between coordinability and dependency constituency. In the examples where both

conjuncts are dependency constituents, namely (25), (26), (28) and (29), coordination is always possible. This suggests that hypothesis 2 might be replaced by the following weaker claim:

Hypothesis 3 *If two (or more) dependency constituents can be given the same type X, then they can be coordinated to give a string of type X.*[5]

But in the examples where at least one conjunct is not a dependency constituent, coordination appears to be restricted to cases like (27), (31) and (33) where the conjuncts are structurally similar; it is not possible for the other examples (30), (32) and (34). In order to discriminate between these cases we need a more precise notion of 'like structure', which we shall discuss in the next subsection.

So far we have not examined the converse claim to hypothesis 2:

Hypothesis 4 *If two (or more) strings can be coordinated to give a string of type X, then each can be given type X.*

In order to maintain this hypothesis we must have some mechanism for dealing with well-known cases of 'unlike category coordination' such as (35):

(35) John is [lucky] and [a rogue].

Whatever the type of ***lucky and a rogue***, it must be assignable to both ***lucky*** and ***a rogue***. If we say that these two items have only the lexical types AdnPre and NP respectively, and that ***is*** is lexically ambiguous between (NP\S)/AdnPre and (NP\S)/NP, then we are forced to conclude that no type is assignable to ***lucky and a rogue***. But if we give ***lucky*** and ***a rogue*** an additional shared lexical type, PredP say, and give ***is*** the single lexical type (NP\S)/PredP, then assigning the type PredP to ***lucky and a rogue*** is consistent with hypothesis 4. Alternatively we might extend **L** with boolean operators, as suggested in Morrill (1990), to achieve a similar effect.

From hypotheses 3 and 4 it follows that two (or more) dependency constituents can be coordinated to give a string of type X iff each has type X.

3.2 Coordination of Non-Dependency Constituents

Hypothesis 3 puts no restrictions on the internal structure of the two dependency constituents that are coordinated, as we can see by comparing (25) and (28), or (26) and (29). But even when the conjuncts themselves are not dependency constituents, there appears to be no restriction on